Technical Presentation **2G Robotics**

Introduction to Non-Linear Optimization for SLAM

Sally Hui

These notes are informally written and may contain errors. Any such errors are unintentional.

1.1 Motivation

Understanding non-linear optimization is fundamental to understanding SLAM. After we have features extracted from images, and sensor measurements from LIDAR, IMU, etc., how do we find trajectory of the robot? Through this technical presentation, you will learn how to formulate and solve a simple non-negative least squares optimization (NNLS) problem.

1.2 Optimization

Throughout this talk we are concerned with the following problem:

$$x^* = \arg\min_x F(x) \tag{1.1}$$

That is, we seek to find the estimation parameter x, within a feasible set, that minimizes the objective function $F(x)$. An objective function is one that maps the parameters onto the set of real numbers. This is the "cost" function associated with the problem.

This means that when we want to pose an optimization problem, we will have to define the parameter(s) of the problem and a cost function to minimize.

1.3 Problem Definition

To demonstrate the techniques involved, a toy example will be presented. The problem to be solved is illustrated (only draw 2 poses):

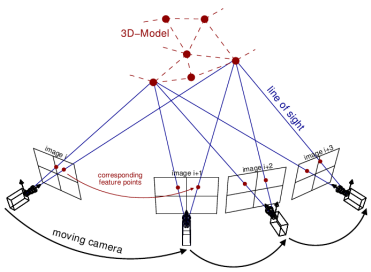

Figure 1.1: http://www.theia-sfm.org/sfm.html

Recall that we refer to an object's position and orientation as its *pose*.

A camera moves between two poses. At each pose, the camera can see a common set of n points, with index *1* to *n*. Assume that they have been triangulated (may be explain it in few words) between the two poses, and are denoted in the frame of the first camera as $Z_1^1 \cdots Z_n^1$. Each of their image coordinates, as observed by the second camera, are denoted $z_1^2 \dots z_n^2$ (may be explain what you mean by upper and lower scripts, and lower and upper case Z). Find the transformation $T^{2:1} \in \mathbf{SE(3)}$ (the estimation parameter) that takes points in the frame of the camera's first pose to the frame of the camera's second pose.

Recall that by $\mathbf{SE(3)}$, we refer to the Special Euclidean group representing rigid body transformations. It is a Lie group and a manifold, with a structure $\mathbf{SO(3)} \times \mathbb{R}^3$.

That is, the transformation we seek has the following structure:

$$T = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \tag{1.2}$$

Here, recall that the group of pure rotations is $R \in \mathbf{SO(3)}$, and the translation is $t \in \mathbb{R}^3$.

To define the cost function we have to define a residual term, or error term. A common one we will use is the reprojection error, which at a point i is given by

$$e_i(z_i, x_i) = z_i^2 - \pi(T^{2:1} Z_i^1) \tag{1.3}$$

We write e_i as a function of z_i and x_i without Z_i, as the 3D points Z_i are assumed to be triangulated from the camera image measurements z_i.

π is the projection function that takes points from 3D space to the camera's 2D image coordinates. We will use a pinhole model for the camera, and assume no distortion. That is, points in 3D space are mapped to the camera's image by the intrinsic matrix

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1.4}$$

and then normalized by dividing by the third coordinate. Recall that f_x and f_y denote the focal length of the camera, which are usually the same, and c_x and c_y denote the center of the camera relative to the origin.

With a slight abuse of notation this gives the projection function $\pi : \mathbb{R}^3 \to \mathbb{R}^2$ defined by the equation

$$\pi\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} (f_x x + c_x z)/z \\ (f_y y + c_y z)/z \end{bmatrix}. \tag{1.5}$$

We have defined a residual. As the cost function, we will use the sum of squared residuals, which is generally stated as:

$$F(x) = \sum_{i=1}^{n} e_i(z_i, x_i)^T \Omega_i e_i(z_i, x_i), \tag{1.6}$$

where Ω is the information matrix associated with the problem, and which we'll leave as identity for now. The interpretation of leaving it as identity is that we will trust all measurements equally (we won't weight them unequally).

That is, our optimization problem is:

$$x^* = \arg\min_x F(x) \tag{1.7}$$

1.4 Gauss-Newton Algorithm

We will now discuss the most common algorithm used to perform least squares minimization, the basic Gauss-Newton method.

As the cost function is non-linear, we will linearize about a linearization point $\breve{x}$ using a first-order Taylor series expansion:

$$\begin{aligned} e_i(\breve{x}_i + \Delta x_i) &= e_i(\breve{x} + \Delta x) \\ &\approx e_i(\breve{x}) + J_i(\Delta x) \end{aligned} \tag{1.8}$$

,

where the J_i is the jacobian of the error term.

Plugging this back into the original cost function,

$$\begin{aligned} F_i(\breve{x} + \Delta x) &= e_i(\breve{x} + \Delta x)^T \Omega_i e_i(\breve{x} + \Delta x) \\ &\approx (e_i(\breve{x}) + J_i \Delta x)^T \Omega_i (e_i(\breve{x}) + J_i \Delta x) \\ &= e_i(\breve{x})^T \Omega_i e_i(\breve{x}) + 2 e_i(\breve{x})^T \Omega_i J_i \Delta x + \Delta x^T J_i^T \Omega_i J_i \Delta x \\ &= c_i + 2 b_i \Delta x + \Delta x^T H_i \Delta x \end{aligned} \tag{1.9}$$

So

$$F(\breve{x} + \Delta x) = c + 2 b^T \Delta x + \Delta x^T H \Delta x \tag{1.10}$$

Where we define c_i, a constant term, as $e_i(\breve{x})^T \Omega_i e_i(\breve{x})$, b_i as $e_i(\breve{x})^T \Omega_i J_i$, and H_i as $J_i^T \Omega_i J_i$.

We can find the parameter values that minimize the cost function by finding the parameter update Δx and applying that to the linearization point $\breve{x}$. To do this we find the derivative of the cost function with respect to Δx, set it to zero, and solve for Δx.

The derivative with respect to the parameter update Δx is:

$$\frac{\partial F(\breve{x} + \Delta x)}{\partial \Delta x} = 0 + 2 b^T + \Delta x^T (H + H^T) \tag{1.11}$$

According to Schwarz's theorem, a Hessian is symmetric if the following holds, for a function f where the second derivative is with respect to two variables x and y:

$$H = \frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x} \tag{1.12}$$

Since in our case the Hessian is the second derivative of only one parameter x, it automatically satisfies Schwarz's theorem.

Then, we have:

$$\frac{\partial F(\breve{x} + \Delta x)}{\partial \Delta x} = 0 + 2 b^T + 2 \Delta x^T H^T \tag{1.13}$$

Setting it to zero,

$$\begin{aligned} 0 &= 2 b^T + 2 \Delta x^T H^T \\ \Delta x^T H^T &= -b^T \\ H \Delta x &= -b \end{aligned} \tag{1.14}$$

This is the update step for the Gauss-Newton algorithm. So to perform the update, we need to find H and b, which are $J^T J$ and $J^T e(\breve{x})$ respectively, if we set Ω to I. How do we find the Jacobian J?

1.5 Jacobians

Recall that the Jacobian is of the error term, where we have entries J_1 to J_n corresponding to each point observed. This means the Jacobian takes the following structure:

$$J = \begin{bmatrix} J_1 \\ J_2 \\ \vdots \\ J_n \end{bmatrix} \tag{1.15}$$

Where each row corresponds to a measured point.

Recall that a function's Jacobian is simply the matrix of all first-order partial derivatives of the function. Since the only parameter we are varying is the transformation between the two poses of the camera and all other variables are assumed constant, there is simply one partial we need to calculate,

$$\frac{\partial e}{\partial T^{2:1}} \tag{1.16}$$

Recall that the residual chosen was

$$e_i(z_i, x_i) = z_i^2 - \pi(T^{2:1} Z_i^1) \tag{1.17}$$

We can use the chain rule:

$$\frac{\partial e}{\partial T^{2:1}} = \frac{\partial e_i}{\partial \pi(T^{2:1} Z_i^1)} \frac{\partial \pi(T^{2:1} Z_i^1)}{\partial T^{2:1} Z_i^1} \frac{\partial T^{2:1} Z_i^1}{\partial T^{2:1}} \tag{1.18}$$

We can now find the expression for each constituent partial derivative. The first one is simply

$$\frac{\partial e_i}{\partial \pi(T^{2:1} Z_i^1)} = -1 \tag{1.19}$$

In the second one, we want to find the partial derivative of the projection function with respect to a 3D point. Recall that the projection function is defined by

$$\pi\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} (f_x x + c_x z)/z \\ (f_y y + c_y z)/z \end{bmatrix} \tag{1.20}$$

We want the Jacobian of this function with respect to the transformed point

$$T^{2:1} Z = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = Z' \tag{1.21}$$

For ease of notation, let π_1 represent $(f_x x + c_x z)/z$ and let π_2 represent $(f_y y + c_y z)/z$.

This means the Jacobian we are finding has the following structure:

$$\frac{\partial \pi(T^{2:1} Z_i^1)}{\partial T^{2:1} Z_i^1} = \frac{\partial \pi(Z')}{\partial Z'} = \begin{bmatrix} \frac{\partial \pi_1}{\partial x} & \frac{\partial \pi_1}{\partial y} & \frac{\partial \pi_1}{\partial z} \\ \frac{\partial \pi_2}{\partial x} & \frac{\partial \pi_2}{\partial y} & \frac{\partial \pi_2}{\partial z} \end{bmatrix} \tag{1.22}$$

This gives:

$$\frac{\partial \pi(Z')}{\partial Z'} = \begin{bmatrix} \frac{f_x}{z} & 0 & \frac{-f_x x}{z^2} \\ 0 & \frac{f_y}{z} & \frac{-f_y y}{z^2} \end{bmatrix} \tag{1.23}$$

Finally, we require

$$\frac{\partial T^{2:1} Z_i^1}{\partial T^{2:1}} \tag{1.24}$$

From [1], we have the following identity:

$$\frac{\partial RZ}{\partial R} = -(RZ)^\times \tag{1.25}$$

Where $R \in \mathbf{SO(3)}$. The proof of that identity is in the appendix of the referenced paper.

We can consider the transformation $T^{2:1}$ as $\mathbf{SO(3)} \times \mathbb{R}^3$. That is,

$$\begin{aligned} T(Z) &= \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \\ &= R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + t \end{aligned} \tag{1.26}$$

The Jacobian of $T(Z)$ with respect to the variables R and t are therefore

$$\begin{aligned} \frac{\partial T^{2:1} Z_i^1}{\partial T^{2:1}} &= \begin{bmatrix} \frac{\partial T(Z)}{\partial R} & \frac{\partial T(Z)}{t} \end{bmatrix} \\ &= \begin{bmatrix} -(RZ)^\times & I \end{bmatrix} \end{aligned} \tag{1.27}$$

1.6 Putting it together

Now we have the jacobians required, we have all the required parameters to compute an update step of the Gauss-Newton algorithm.

$$H \Delta x = -b \tag{1.28}$$

We can compute H by $J^T J$, and b by $J^T e(\breve{x})$.

So, to reiterate, our algorithm is:

1. Initialize the transformation $T^{2:1}$

2. Calculate H by $J^T J$
3. Calculate b by $J^T e$
4. Calculate $\Delta x = -H^{-1} b$
5. Apply Δx to the current estimate of x.
6. Check for some stopping condition (e.g. the norm of Δx is smaller than a certain threshold). If the condition is not reached, perform steps 2-6 again. Else, we have found a candidate $T^{2:1}$.

References

[1] Michael Bloesch, Hannes Sommer, Tristan Laidlow, Michael Burri, Gabriel Nuetzi, Pter Fankhauser, Dario Bellicoso, Christian Gehring, Stefan Leutenegger, Marco Hutter, Roland Siegwart. *A Primer on the Differential Calculus of 3D Orientations.*, `https://arxiv.org/pdf/1606.05285.pdf`